Insecure Dependencies

Common Mistakes

- Not knowing used version of dependencies (client- and server-side)
  - Includes directly used components *and* nested dependencies
- **Used software is vulnerable, unsupported, or out of date**
  - Includes OS, Web/Application server, DBMS, Applications, APIs, components, runtime environments and libraries
- Lack of regular scans and bulletin subscriptions for vulnerabilities
- **Patch Management Process insufficient** or missing
- Component configuration not secured (see Insecure Configuration)

Variable Exploitability

- Already-written exploits for many known vulnerabilities exist
- Other vulnerabilities require a custom exploit (e.g. Zero Days)

Variable Impact

- Full range of weaknesses is possible
  - RCE, Injection, Broken Access Control, XSS, etc.
- Impact could be minimal, up to host takeover and data compromise

ℹ️ *Some of the largest breaches to date have relied on exploiting known vulnerabilities in components (e.g. Equifax)*

Data Factors

A06:2021 – Vulnerable and Outdated Components

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
3	27.96%	8.77%	51.78%	22.47%	5.00	5.00	30,457	0

Prevention

- **Removing unused dependencies, unnecessary features, components, files, and documentation** 💯
- **Continuously inventorying the versions of** both client-side and server-side **components** (e.g., frameworks, libraries) **and their dependencies** using tools like OWASP Dependency Check, retire.js, etc.
- Continuously monitoring sources like Common Vulnerability and Exposures (CVE) and National Vulnerability Database (NVD) for vulnerabilities in the components
- Using software composition analysis tools to automate the process
- Subscribing to email alerts for security vulnerabilities related to components you use

- Obtain components from official sources over secure links only
- Preferring signed packages to reduce the chance of including a modified, malicious component (see Software and Data Integrity Failures).
- Monitor for libraries and components that are unmaintained or do not create security patches for older versions
- If patching is not possible, deploying a virtual patch to monitor, detect, or protect against the discovered issue is an option

⚠️ *Every organization must ensure an ongoing plan for monitoring, triaging, and applying updates or configuration changes for the lifetime of the application or portfolio.*

OWASP Dependency-Check

> **Dependency-Check is a utility that identifies project dependencies and checks if there are any known, publicly disclosed, vulnerabilities.** Currently, Java and .NET are supported; additional experimental support has been added for Ruby, Node.js, Python, and limited support for C/C++ build systems (autoconf and cmake). The tool can be part of a solution to the OWASP Top 10 2017 A9:2017-Using Components with Known Vulnerabilities.

Exercise 7.1 (🏠)

1. Open the Sample Report of OWASP Dependency-Check
2. Recommend an action plan for the assessed application
    - Distinguish between *short term* and *long term* actions
3. Collect potential risks for a corporate-wide rollout of OWASP Dependency-Check in your organization
4. Collect possible mitigations for those risks

💡 *If your organization does not use any technology that could be assessed by OWASP Dependency-Check, simply perform the exercise pretending it actually could.*

Exercise 7.2

1. Download the `/ftp/package.json.bak` from your local Juice Shop
2. Rename it into `package.json` and run `npm audit`. Follow the advise given in case of an error and then rerun it
3. Check any subsequent error messages and get their root cause out of the way so you can run the audit successfully.
4. After `npm audit` succeeds, read up on the listed vulnerabilities and use the most promising findings during Exercise 7.4

💡 *After completing Exercise 7.4 your could also run `npm audit` on the (normally off-limits) `package.json` and `/frontend/package.json` files…*

Insecure Configuration

Common Mistakes

- **Missing appropriate security hardening** across application stack
- Improperly configured permissions on cloud services
- **Unnecessary features are enabled or installed**
- **Default accounts** and their passwords **still enabled** and unchanged
- Overly informative error messages (e.g. revealing stack traces)
- Latest security features are disabled or not configured securely
- No proper security headers or directives configured on server
- Software is out of date or vulnerable (see Insecure Dependencies)

Potential Impact

- Unauthorized access to
  - system data
  - functionality
- Complete system compromise
  - e.g. by installing a web shell or backdoor

ℹ️ *Especially information leakage from error message can often be useful to attackers in subsequent attacks like Injection or Privilege Escalation.*

Data Factors

A05:2021 – Security Misconfiguration

CWEs Mapped	Max Incidence Rate	Avg Incidence Rate	Avg Weighted Exploit	Avg Weighted Impact	Max Coverage	Avg Coverage	Total Occurrences	Total CVEs
20	19.84%	4.51%	8.12	6.56	89.58%	44.84%	208,387	789

Web Shells

> **A web shell is a script that can be uploaded to a web server to enable remote administration of the machine.** Infected web servers can be either Internet-facing or internal to the network, where the web shell is used to pivot further to internal hosts.
>
> **A web shell can be written in any language that the target web server supports.** The most commonly observed web shells are written in languages that are widely supported, such as PHP and ASP. Perl, Ruby, Python, and Unix shell scripts are also used. [^1]

> Using network reconnaissance tools, **an adversary can identify vulnerabilities that can be exploited and result in the installation of a web shell.** For example, these vulnerabilities can exist in content management systems (CMS) or web server software.
>
> Once successfully uploaded, **an adversary can use the web shell to leverage other exploitation techniques to escalate privileges and to issue commands remotely.** These commands are directly linked to the privilege and functionality available to the web server and may include the ability to add, delete, and execute files as well as the ability to run shell commands, further executables, or scripts. [^1]

Shodan

> **Shodan is a search engine for Internet-connected devices.** Web search engines, such as Google and Bing, are great for finding websites. But what if you're interested in measuring which countries are becoming more connected? Or if you want to know which version of Microsoft IIS is the most popular? Or you want to find the control servers for malware? Maybe a new vulnerability came out and you want to see how many hosts it could affect? Traditional web search engines don't let you answer those questions. [^2]

> Shodan gathers information about all devices directly connected to the Internet. **If a device is directly hooked up to the Internet then Shodan queries it for various publicly-available information.** The types of devices that are indexed can vary tremendously: ranging from small desktops up to nuclear power plants and everything in between.
>
> So what does Shodan index then? **The bulk of the data is taken from *banners*, which are metadata about a software that's running on a device**. This can be information about the server software, what options the service supports, a welcome message or anything else that the client would like to know before interacting with the server. [^2]

Areas of application for Shodan services

> - **Network Security**: keep an eye on all devices at your organization that are facing the Internet
> - **Market Research**: find out which products people are using in the real-world
> - **Cyber Risk**: include the online exposure of your vendors as a risk metric
> - **Internet of Things**: track the growing usage of smart devices
> - **Tracking Ransomware**: measure how many devices have been impacted by ransomware [^2]

Shodan Query Examples

❌ Search in Banner Data ("Google-style")

```
nordakademie
# = results with "nordakademie" in banner text
```

Search in Meta Data with `filtername:value`

```
hostname:nordakademie
# = results with "nordakademie" in host name

hostname:nordakademie product:apache -hash:0
# = Apache servers answering with non-empty banners

hostname:nordakademie http.component:php http.status:200
# = available (200 "OK") servers running PHP
```

Filtering for known vulnerabilities by CVE-ID

```
http.component:php vuln:CVE-2018-10547
# = Servers running PHP vulnerable to Reflected XSS on some error pages
```

⚠️ *The `vuln` filter is only available to academic users or Small Business API subscription and higher.*

REST and Streaming API

see https://developer.shodan.io/api

Literature Recommendations *(optional)*

- Matherly: Complete Guide to Shodan, 2017

Exercise 7.3

1. Get an idea of the available Shodan query filters on https://developer.shodan.io/api/banner-specification
2. Perform some Shodan searches for Internet-connected devices associated with your current employer (e.g. by `hostname` or `ip`)

---

3. Use Mozilla Observatory to check the rating of at least one SSL configuration of your employer's hosts

ℹ️ *Please consider ticking the "Don't include my site in the public results" checkbox before running the online scan!*

Prevention

- **Establishing a repeatable hardening process**
- **Configuring development, QA, and production environments identically** (but with different credentials!)
- Operating a minimal platform without any unnecessary features, components, documentation, and samples
- Removing or not installing unused features and frameworks
- Reviewing and Updating the configurations appropriate to all security notes, updates, and patches as part of the patch management process (see Insecure Dependencies)

- Use a segmented application architecture for effective and secure separation between components or tenants, with segmentation, containerization, or cloud security groups (ACLs)
- Sending security directives to clients, e.g., Security Headers
- Establishing an automated process to verify the effectiveness of the configurations and settings in all environments

Exercise 7.4 (🏠)

1. Persist a Stored XSS attack via the *Contact Us* page (⭐⭐⭐⭐)
2. Report the vulnerability which makes this XSS possible (⭐⭐⭐⭐)

ℹ️ *To report anything to the shop, you can use the "Customer Feedback" page. You have to supply as detailed information as possible.*

Exercise 7.5 (🏠)

1. Visit the *Support Chat* page and talk to the friendly bot
2. Try asking it for product prices, deluxe membership our coupons
3. Bully the chatbot until it gives you a coupon code (⭐)
4. Analyze the source code of the chatbot and kill it (⭐⭐⭐⭐⭐)

XXE

(XML External Entities)

XML Entities

- In the Document Type Definition (DTD) you specify shortcuts as `ENTITY`...
  - `<!ENTITY author "Bjoern Kimminich">`
  - `<!ENTITY copyright "(C) 2018">`
- ...to later dereference them in the XML
  - `<author>&author; &copyright;</author>`

External Entities

- DTD changed to use External Entities...
  - `<!ENTITY author SYSTEM "https://raw.githubusercontent.com/juice-shop/juice-shop/gh-pages/entities.dtd">`
  - `<!ENTITY copyright SYSTEM "https://raw.githubusercontent.com/juice-shop/juice-shop/gh-pages/entities.dtd">`
- ...whereas the XML stays the same
  - `<author>&author; &copyright;</author>`

Attack Vector XXE

- Many older or poorly configured XML processors evaluate external entity references within XML documents
- External entities can be abused for
  - disclosure of internal files
  - internal port scanning
  - remote code execution
  - denial of service attacks

XML with Attack Payloads

Extracting Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]>
    <foo>&xxe;</foo>
```

Network Probing

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "https://192.168.1.1/private" >]>
    <foo>&xxe;</foo>
```

DoS Attack (against Linux-based Systems)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
    <!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///dev/random" >]>
    <foo>&xxe;</foo>
```

Exercise 7.6

1. Identify the weak point of the application that accepts arbitrary XML data as input (★★)
2. Retrieve the content of your local system's `C:\Windows\system.ini` (or `/etc/passwd` if you are using Linux) via an XXE attack (★★★)

Prevention

- **Configure XML parser to**
  - **disable DTDs completely** (by disallowing `DOCTYPE` declarations) 💯
  - disable External Entities (only if allowing DTDs cannot be avoided)

❌ Selective validation or escaping of tainted data is **not** sufficient, as the whole XML document is crafted by the attacker!

XML Parser Hardening Examples

`libxml2` (C/C++)

- `XML_PARSE_NOENT` and `XML_PARSE_DTDLOAD` must **not be defined** in the Enum `xmlParserOption`.

ℹ️ *Starting with release `2.9` entity expansion is disabled by default. Using any older version makes it more likely to have XXE problems if the configuration was not explicitly hardened.*

`org.dom4j.io.SAXReader` (Java)

```
saxReader.setFeature(
  "http://apache.org/xml/features/disallow-doctype-decl", true);
saxReader.setFeature(
  "http://xml.org/sax/features/external-general-entities", false);
saxReader.setFeature(
  "http://xml.org/sax/features/external-parameter-entities", false);
```

`java.beans.XMLDecoder` (Java)

- The `readObject()` method in this class is fundamentally unsafe
- It is vulnerable against XXE as well as arbitrary code execution
- There is no way to make use of this class safe

⚠️ *Most Java XML parsers have insecure parser settings by default!*

Exercise 7.7 *(optional)*

1. Perform a DoS-like Attack using XXE (★★★★★)
2. Read up on all vulnerabilities associated with JWT from the `npm audit` of the Juice Shop's leaked `/ftp/package.json.bak`
3. Forge an essentially unsigned JWT token (★★★★★)
4. Forge an almost properly RSA-signed JWT token (★★★★★★)